SISTEM PENILAI SOURCE CODE OTOMATIS

Meylanie Olivya
Politeknik Negeri Ujung Pandang
livya.me@gmail.com

ABSTRAK

Penelitian ini bertujuan membangun sistem yang dapat menganalisis source code program dalam dua bahasa pemrograman yaitu Pascal dan Java, dan kemudian mengimplementasikannya sehingga dapat digunakan untuk memeriksa tugas source code program mahasiswa.

Metode penelitian yang digunakan adalah eksperimental, dimana penelitian dilakukan dengan melakukan pemeriksaan terhadap tugas mahasiswa dalam tiga kelas yang berbeda, dua kelas pemrograman Pascal dan satu kelas pemrograman Java. Tiap kelas terdiri dari 30-35 mahasiswa. Banyaknya kasus yang digunakan adalah tiga kasus. Baik mahasiswa maupun dosen memasukkan source code jawaban melalui Web Browser.

Hasil penelitian menunjukkan bahwa sistem dapat memeriksa source code dalam bahasa pemrograman Pascal maupun Java. Selain itu, banyaknya waktu yang digunakan oleh sistem untuk memeriksa tugas source code dalam satu kelas jauh lebih singkat dibanding waktu kerja manual, sehingga dapat dikatakan bahwa sistem mampu meningkatkan efisiensi kerja dosen.

1. PENDAHULUAN

Mata kuliah pemrograman dasar di banyak universitas pada umumnya diikuti oleh banyak mahasiswa. Dengan semakin besarnya jumlah mahasiswa, semakin banyak pula tugas pemrograman dalam bentuk *source code* yang harus diperiksa. Pemeriksaan ini akan membutuhkan waktu dan tenaga yang besar pula dan kadang melebihi yang dihabiskan selama interaksi langsung. Oleh karena itu, otomasi penilaian terhadap tugas *source code* telah menjadi suatu kebutuhan penting.

Di dalam lingkungan Universitas Hasanuddin sendiri, juga telah dibangun sistem penilai otomatis. Sistem yang pertama dibangun oleh Lidemar Halide. Sistem ini menilai program hanya berdasarkan pengujian terhadap output program. Sistem yang kedua dibangun oleh Rani Purbaningtyas, yang menilai program berdasarkan dua kategori, yaitu pengujian terhadap input-output program, dan pengujian terhadap *source code* program. Pada sistem yang kedua, mahasiswa diharuskan menggunakan kode operasi yang telah ditentukan oleh dosen pengampu, sehingga mahasiswa tidak dapat berkreasi. Kedua sistem di atas dibangun untuk mata kuliah pemrograman dengan bahasa Java.

Meskipun memiliki kekurangan tersendiri dan masih dianggap belum mampu menangani proses grading secara utuh, keberadaan penilai otomatis banyak membantu untuk menangani kelas yang besar. Keuntungan utama penggunaannya adalah pada penghematan waktu yang dibutuhkan untuk memeriksa *source code* – pada satu kasus hingga melebihi 85% [Chi, 1998]. Dengan bantuan penilai otomatis, diharapkan instruktur lebih dapat berkonsentrasi pada interaksi dengan mahasiswanya lewat kuliah atau diskusi, dan mahasiswa terpacu untuk mengerjakan tugasnya dengan lebih baik.

2. TINJAUAN PUSTAKA

2.1 Computer Assisted Assessment

Computer Assisted Assessment (CAA) merupakan istilah umum untuk penggunaan teknologi komputer dalam melakukan penilaian, termasuk dalam tugas dan ujian [Rawles, 2002]. Suatu perangkat lunak CAA dapat mengotomatisasi aspek-aspek tertentu pada penilaian, yang dikategorikan secara luas sebagai perangkat untuk mengadakan, memberi nilai, dan memberikan umpan balik. Namun, dalam penelitian ini, perangkat lunak CAA yang dibangun akan dibatasi secara lebih spesifik, yaitu sebagai pemberi nilai dan juga dapat memberikan umpan balik.

2.2 Metodologi Penilaian Program

Sebuah program dapat dinilai menggunakan dua pendekatan : pendekatan *black box* dan pendekatan *white box*. Pada pendekatan *black box*, sebuah program dianggap sebagai "kotak hitam" yang dinilai hanya menggunakan masukan dan keluaran program tersebut. Pada pendekatan *white box*, sebuah program dinilai berdasarkan kualitas intrinsik dari program tersebut. Penilaian program dapat dilakukan menggunakan salah satu pendekatan atau kombinasi dari keduanya.

2.2.1 Pendekatan Black Box

Sebuah program adalah sebuah spesifikasi untuk proses komputasi [Aaby, 2004]. Dengan kata lain, program komputer dibuat agar komputer dapat melakukan komputasi untuk memenuhi tujuan tertentu. Oleh karena itu, sebuah program dikatakan tepat (*correct*) apabila komputasi yang dilakukan program tersebut sesuai dengan tujuan pembuatannya. Penilaian terhadap sebuah program pertama-tama dilakukan terhadap ketepatannya, karena program yang tidak tepat dapat dikatakan gagal memenuhi tujuannya.

Dalam pendekatan *black box*, proses penilaian dilakukan dalam dua tahap, tahap analisis statik dan tahap analisis dinamik. Dalam tahap analisis statik, *source code* program diubah ke dalam format siap eksekusi menggunakan kompilator atau interpreter. Dalam tahap analisis dinamik, program dieksekusi, diberi masukan dan kemudian keluaran yang dihasilkannya diperiksa.

2.2.2 Pendekatan White Box

Dalam pendekatan *white box*, penilaian program dilakukan terhadap kualitas program menurut bagaimana dan seberapa baik program memecahkan sebuah masalah. Seorang penilai manusia dapat dengan mudah menilai pemecahan masalah yang diberikan dalam bentuk program, namun sebuah penilai otomatis tidak dapat melakukan hal ini secara penuh. Sebuah penilai otomatis dapat menilai kualitas program secara parsial dengan menggunakan besaran-besaran perangkat lunak (software metrics). Penggunaan besaran perangkat lunak sebagai bagian dalam proses penilaian ini dipelopori oleh ASSYST [Blumenstein, 2004].

Untuk sebuah proyek perangkat lunak kecil seperti tugas praktikum atau ujian, Burgess [Zin, 2001] menyatakan bahwa hanya empat faktor kualitas yang perlu dipertimbangkan: ketepatan (correctness), kemudahan perawatan (maintainability), kemudahan penggunaan (usability) dan efisiensi. Faktor kemudahan perawatan/maintainability umumnya diwakili oleh kompleksitas dan tipografi kode. Ketepatan, kompleksitas, keterbacaan atau tipografi *source code*, dan efisiensi telah digunakan oleh berbagai aplikasi penilai otomatis sebagai komponen-komponen penilaian [Carter, 2003].

2.3 Teknik Pemrosesan Source Code

2.3.1 Analisis Leksikal

Analisis leksikal padalah proses pengubahan teks kode program menjadi serangkaian simbol yang valid menurut grammar (tata bahasa) yang bersangkutan. Program yang melakukan analisis leksikal disebut sebagai lexer, sementara simbol-simbol yang dihasilkan oleh lexer ini sering disebut sebagai token. Token-token ini akan dijadikan masukan untuk proses selanjutnya, yaitu proses analisis sintaks. Token-token yang umum misalnya kata kunci/keyword, tanda baca, atau identifier (nama variabel, konstanta, fungsi dan prosedur).

Lexer pada umumnya bekerja menggunakan prinsip otomata nondeterministik. Pola leksikal dari token-token umumnya didefinisikan menggunakan regular expression (regex), karena format regex dapat menggambarkan pola leksikal secara kompak. Berikut ini adalah contoh pola regex untuk mengenali identifier (identifier didefinisikan sebagai string yang diawali oleh alfabet dan diikuti oleh sembarang alfabet atau angka) :

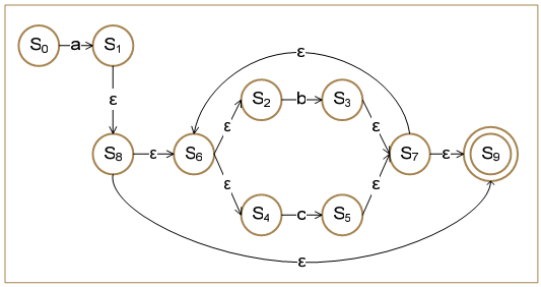
alfabet → (a|b|c|d|e|f|g|h|i|jk|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z)

angka → (0|1|2|3|4|5|6|7|8|9)

identifier→ alfabet(alfabet|angka)*

Dengan menggunakan definisi regex untuk setiap konstruksi token yang mungkin, maka dapat dibentuk sebuah mesin nondeterministic finite automata (NFA) yang akan mengenali token-token pada *source code*. Pada Gambar 1 digambarkan sebuah diagram NFA yang akan mengenali regular expression a(b|c)*.

Alat bantu untuk pembuatan lexer sudah banyak tersedia, di antaranya Lex (generator lexer yang umum digunakan, menghasilkan lexer dalam bahasa C) / Flex (versi open source dari Lex) / JFlex (generator lexer untuk bahasa Java), atau re2c.



Gambar 1 Diagram NFA untuk pola regular expression a(b|c)* [Cooper, 2003]

2.3.2 Analisis Sintaksis

Analisis sintaks, yang juga dikenal sebagai *parsing*, adalah proses verifikasi terhadap *source code* untuk menguji apakah program yang dihasilkan merupakan kalimat yang legal menurut aturan sintaks / grammar pada bahasa yang bersangkutan. Aturan sintaks atau aturan tata bahasa yang berlaku untuk bahasa alami maupun bahasa pemrograman secara formal didefinisikan oleh Noam Chomsky [Grune, 1998]. Menurut Chomsky, sebuah tata bahasa / grammar <Σ, N, P, S> terdiri dari empat bagian :

1). Sebuah himpunan terhingga Σ yang terdiri dari simbol terminal, komponen penyusun terkecil dalam bahasa tersebut.
2). Sebuah himpunan terhingga N yang terdiri dari simbol nonterminal atau kategori sintaks, yang masing-masing merupakan koleksi subfrase dari kalimat.
3). Sebuah himpunan terhingga P yang terdiri dari aturan atau produksi yang menggambarkan bagaimana tiap simbol nonterminal didefinisikan menggunakan simbol terminal dan/atau simbol nonterminal.
4). Sebuah simbol nonterminal istimewa S, simbol awal, yang menspesifikasikan entitas utama yang didefinisikan – misalnya kalimat atau program.

Dalam konteks bahasa pemrograman, simbol terminal adalah token-token yang dikenali dalam bahasa tertentu. Simbol nonterminal bersama dengan aturan-aturan produksi

digunakan untuk mendefinisikan bagaimana menyusun simbol-simbol terminal menjadi kalimat-kalimat yang valid agar dapat dipahami. Simbol awal merepresentasikan entitas yang hendak dibangun – misalnya, simbol awal dalam bahasa Pascal adalah <program>.

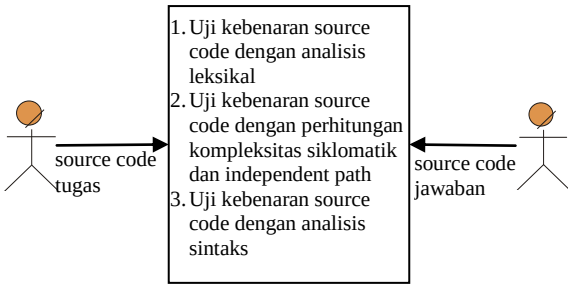
3. ANALISIS DAN DESAIN SISTEM

Jenis penelitian yang dilakukan adalah penelitian yang bersifat eksperimental, yaitu membangun sebuah perangkat lunak yang dapat menilai secara otomatis *source code* dalam berbagai bahasa pemrograman. Setelah itu, perangkat lunak yang telah dibangun akan diujicobakan terhadap tugas-tugas mahasiswa dalam mata kuliah yang berbasis pemrograman.

Sistem yang akan dibangun adalah sistem yang mampu menganalisis dan menilai *source code* program tanpa melakukan kompilasi terhadap source code tersebut, sehingga dapat digunakan untuk berbagai bahasa pemrograman. Gambaran umum sistem ditunjukkan pada Gambar 2.

Source code akan dinilai secara *black box* maupun *white box*. Penilaian secara *black box* dilakukan berdasarkan analisis statik, yaitu penilaian terhadap sintaks pemrogramannya. Penilaian terhadap sintaks pemrograman source code akan melalui dua proses yaitu analisis leksikal dan analisis sintaks.

Penilaian secara *white box* dilakukan dengan membentuk graf kontrol dari *source code*, dan kemudian menghitung kompleksitas siklomatik dan jumlah independent path-nya. Nilai dari kompleksitas siklomatik dan jumlah independent path akan menjadi parameter penilaian secara *white box*.



Gambar 2 Gambaran umum rancangan sistem penilaian source code otomatis bersifat generic

3.1 Identifikasi Kebutuhan Sistem

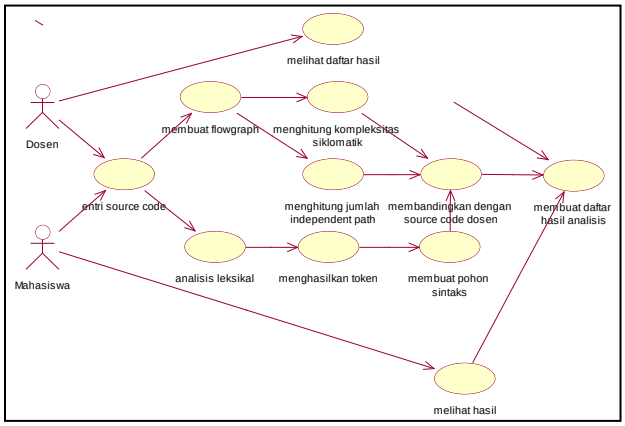
Hasil analisis identifikasi kebutuhan user terhadap sistem penilaian source code otomatis bersifat generik adalah sebagai berikut:

1) User (dosen) memberikan tugas kepada mahasiswa berupa pembuatan source code program yang bertujuan untuk menyelesaikan suatu kasus. Mahasiswa kemudian mengumpulkan tugas tersebut dalam bahasa pemrograman yang telah ditentukan sebelumnya.
2) User (dosen) dapat menggunakan sistem untuk memeriksa tugas pemrograman berbahasa Pascal maupun Java dengan cara memasukkan source code jawaban sebagai source code pembanding terhadap source code mahasiswa.
3) User (dosen dan mahasiswa) dapat memasukkan source code melalui Web Browser.
4) User (dosen) mengharapkan sistem dapat memeriksa kebenaran source code mahasiswa dengan cara melakukan analisis leksikal terhadap source code tersebut.
5) User (dosen) mengharapkan sistem dapat memeriksa kebenaran logika source code mahasiswa dengan cara melakukan perhitungan kompleksitas siklomatik dan jumlah independent path.
6) User (dosen) mengharapkan sistem dapat memeriksa kebenaran sintaks source code

dan juga dapat menilai kemampuan bahasa pemrograman mahasiswa dengan cara melakukan analisis sintaks terhadap source code mahasiswa.

7) User (dosen) mengharapkan keluaran dari sistem ini adalah berupa daftar hasil analisis kebenaran source code.
8) User (dosen dan mahasiswa) mengharapkan bahwa keluaran dari sistem ini dapat ditampilkan melalui Web Browser.

Hasil analisis identifikasi kebutuhan user terhadap sistem penilaian source code otomatis bersifat generik dapat digambarkan dalam bentuk diagram use case seperti yang ditunjukkan pada Gambar 3.



Gambar 3 Diagram use case sistem

4.HASIL PENELITIAN DAN PEMBAHASAN

4.1 Implementasi Sistem

Sistem penilaian source code otomatis bersifat generik terdiri atas dua modul yaitu modul antarmuka dan modul analyzer.

User (dosen dan mahasiswa) memasukkan input source code maupun melihat hasil analisis source code melalui modul antarmuka. Halaman yang digunakan untuk memasukkan source code ditunjukkan pada Gambar 4. Pada halaman ini juga dimasukkan keterangan mengenai kode dosen dan kode soal.

Gambar 4 Contoh pengisian form entri source code

Data source code yang masuk kemudian akan diproses pada modul analyzer yang akan dibahas pada subbab selanjutnya. Jika source code telah dianalisis maka hasilnya akan tampil pada halaman hasil analisis. User akan memasukkan NIM dan setelah itu akan tampil hasil analisis dalam bentuk tabel berdasarkan NIM yang dimasukkan sesuai dengan yang ditunjukkan pada Gambar 5. Sedangkan detail untuk jumlah simpul ditunjukkan pada Gambar 6.

Gambar 5 Tampilan hasil analisis berdasarkan NIM

Gambar 6 Tampilan form detail hasil analisis khusus
detail simpul

4.2 Analisis Pengujian Kebenaran Source Code

Analisis pengujian kebenaran source code melalui tiga tahap berikut ini:

1) Mengecek apakah terdapat error pada source code berdasarkan analisis leksikal
2) Mengecek apakah terdapat error dengan melakukan perbandingan nilai kompleksitas siklomatik dan jumlah independent path pada flowgraph yang dibuat berdasarkan source code.
3) Mengecek apakah terdapat error berdasarkan analisis sintaks.

Berikut adalah contoh kasus pemeriksaan source code berbahasa Pascal. Dosen memberikan tugas kepada mahasiswa untuk membuat program yang dapat mengidentifikasi bonus karyawan. Karyawan mendapatkan bonus jika termasuk golongan 3, dimana hanya terdapat 3 golongan yaitu 1, 2, dan 3. Jika karyawan termasuk golongan 1 atau 2, maka bonusnya 0.

Tabel 1 Contoh source code dosen dan mahasiswa

Source code dosen / jawaban	Source code mahasiswa: Kornales Benamen (081388)

Program Soal3;
var bonus: Integer;
gol: String;
begin
write('Masukkan golongan: ');
readln(gol);
bonus := 0;
if (gol = '3') then
begin
bonus := 100000;
end;
 	
writeln(IntToStr(bonus));
readln;
end.
	Program Soal3;
	var bonus: Integer;
	golongan: String;
	begin
	write('Masukkan golongan: ');
	readln(golongan);
	bonus := 0;
	if (golongan = '3') then
	begin
	bonus := 100000;
	end;
	
	writeln(IntToStr(bonus));
	readln;
	end.

4.2.1 Analisis Leksikal

Pada analisis leksikal, dilakukan pembacaan karakter source code dan memproduksi barisan token. Baris source code yang merupakan ruang kosong seperti karakter spasi, tabulasi dan baris baru, akan dibuang sehingga menghasilkan source code seperti pada Tabel 2.

Tabel 2 Source code yang perintah kosongnya telah dibuang

Source code dosen / jawaban	Source code mahasiswa: Kornales Benamen (081388)

write('Masukkan golongan: ');
readln(gol);
bonus := 0;
if (gol = '3') then
begin
bonus := 100000;
end;
writeln(IntToStr(bonus));
	write('Masukkan golongan: ');
	readln(golongan);
	bonus := 0;
	if (golongan = '3') then
	begin
	bonus := 100000;
	end;
	writeln(IntToStr(bonus));

Setiap token yang diproduksi harus memenuhi tipe token yang sesuai dengan daftar tipe token pada Tabel 3. Daftar tipe token dibuat berdasarkan token-token yang sesuai dengan aturan bahasa pemrograman.

Tabel 3 Daftar tipe token

Tipe Token	Contoh Token
RESERVE_WORD	if, else, while, for, do, to, begin, end, etc
id	nilai, s_nilai, input
num	60, 60.5
equalop	:=
relop	<, >, <>, >=, <=, =
addop	+, -
mulop	* , /
string	'….'
semicolon	;
comma	,
leftbracket	(
rightbracket	)
mleftbracket	[
mrightbracket	]
bleftbracket	{
brightbracket	}

Misalkan untuk baris source code:

if (gol = '3') then

'if'→diidentifikasi sebagai token tipe 'if'

'('→diidentifikasi sebagai token tipe 'leftbracket'

'gol'→diidentifikasi sebagai token tipe 'id'

'='→diidentifikasi sebagai token tipe 'relop'

''3''→diidentifikasi sebagai token tipe 'string'

')'→diidentifikasi sebagai token tipe 'rightbracket'

'then'→ diidentifikasi sebagai token tipe 'then'

Jika semua token pada source code dapat diidentifikasi ke dalam tipe token sesuai dengan Tabel 3, maka analisis pengujian kebenaran source code dapat dilanjutkan ke tahap berikutnya. Tabel 4 menunjukkan contoh barisan token yang diproduksi dari source code dosen. Jika terdapat token dengan tipe yang tidak terdaftar pada Tabel 3, maka dapat disimpulkan bahwa source code tersebut tidak benar. Jika source code tidak benar, maka sistem akan menghasilkan keluaran berupa notifikasi bahwa terdapat variable yang tidak diketahui dan kemudian akan dimasukkan ke dalam daftar hasil analisis. Jika terdapat error pada source code, maka analisis pengujian tidak dapat berlanjut ke tahap berikutnya.

Tabel 4 Contoh barisan token dari source code mahasiswa

Baris	Contoh Token
1	write | leftbracket | 'Masukkan golongan: ' | rightbracket | semicolon
2	readln | leftbracket | id | rightbracket | semicolon
3	id | equalop | 0 | semicolon
4	if | leftbracket | id | relop | '3' | rightbracket | then
5	begin
6	id | equalop | 100000 | semicolon
7	end | semicolon
8	writeln | leftbracket | IntToStr | leftbracket | id | rightbracket | rightbracket | semicolon

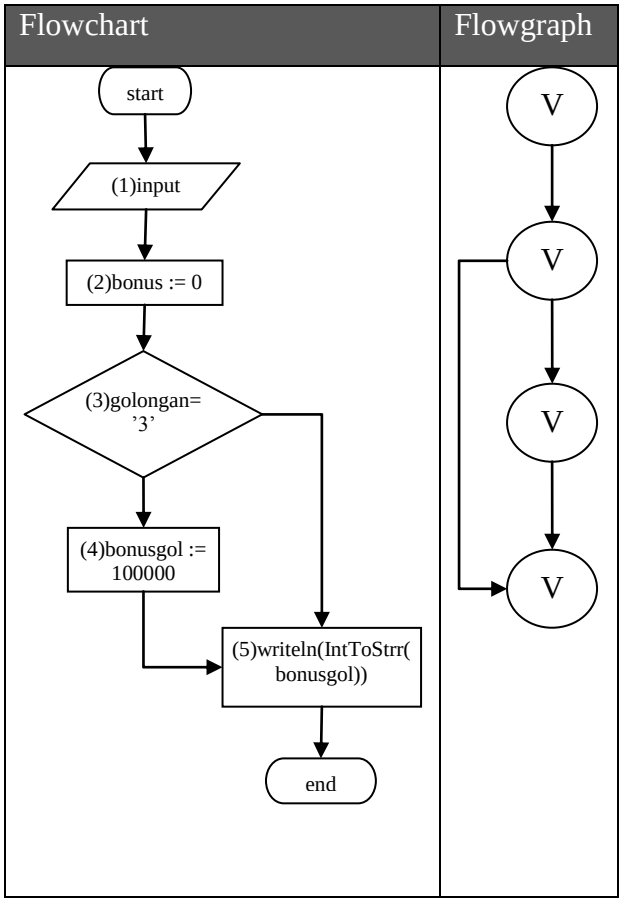
4.2.2 Analisis Flowgraph

Analisis flowgraph adalah tahap kedua yang dilakukan pada analisis pengujian kebenaran source code. Flowgraph dibuat berdasarkan flowchart yang dihasilkan dari source code pada Tabel 2. Gambar 7 menunjukkan flowgraph yang dihasilkan dari source code mahasiswa. Berdasarkan flowgraph tersebut,

kemudian dilakukan perhitungan kompleksitas siklomatik, region, dan jumlah independent path.

Kompleksitas siklomatik = Jumlah sisi – jumlah simpul + 2

Region = Jumlah split node + 1



Gambar 7 Flowchart dan flowgraph dari source code mahasiswa

Berdasarkan Gambar 7 maka dapat ditentukan simpul, sisi, kompleksitas siklomatik, region, dan independent path. Tabel 5 menunjukkan keseluruhan parameter tersebut. Jika kompleksitas siklomatik, region dan jumlah independent path bernilai sama, maka dapat disimpulkan bahwa logika source code adalah benar sehingga dapat dilakukan tahap analisis berikutnya. Jika nilai kompleksitas siklomatik, region, dan jumlah independent path tidak sama, maka dapat disimpulkan bahwa source code tidak benar dan tidak dapat dilakukan tahap analisis selanjutnya. Keluaran untuk tahap analisis flowgraph adalah berupa daftar seperti yang terdapat pada Tabel 5.

Tabel 5 Penentuan parameter flowgraph

Parameter	Detail
Simpul	V0 : (1), (2) V1 : (3) V2 : (4) V3 : (5)
Sisi	V0 → V1 V1 → V2 V1 → V3 V2 → V3
Kompleksitas Siklomatik	Jumlah sisi – Jumlah Simpul + 2 = 4 – 4 + 2 = 2
Region	Jumlah split node + 1 = 1 + 1 = 2
Independent Path	V0 –> V1 –> V2 –> V3 V0 –> V1 –> V3 Maka jumlah path = 2

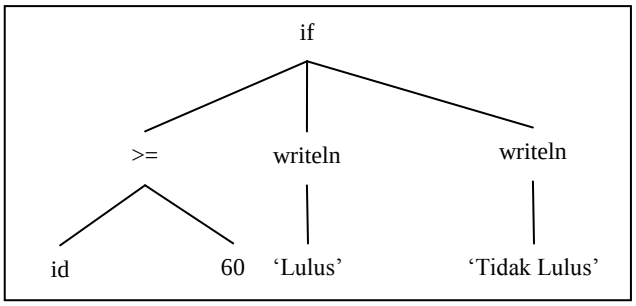
4.2.3 Analisis Sintaks

Pada bahasa pemrograman Pascal maupun Java, setiap baris perintah program harus diakhiri dengan semicolon (;). Namun, untuk perintah yang mengandung logika 'if' dan disertai oleh logika 'else', maka seluruh perintahnya adalah satu baris program. Analisis sintaks membentuk pohon sintaks berdasarkan satu baris program. Misalkan terdapat satu baris program sebagai berikut.

```
if (nilai >= 60) then
begin
    writeln('Lulus');
end;
```

else writeln(‘Tidak Lulus’);

Dari sintaks di atas, akan dibentuk pohon sintaks sebagai berikut.



Gambar 8 Contoh pohon sintaks

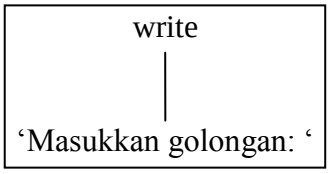
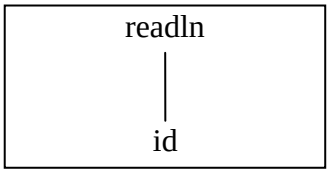
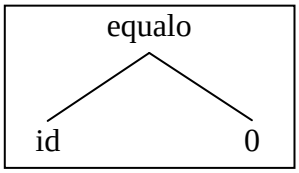
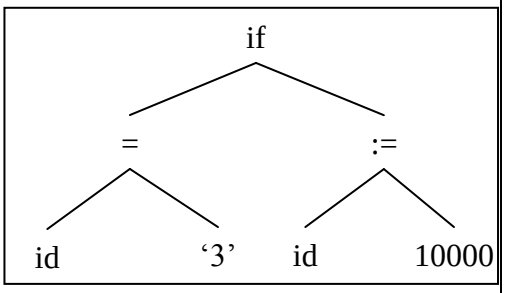
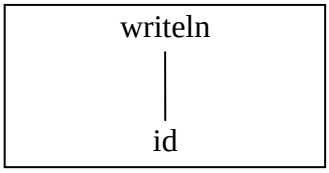
Simpul di sebelah kiri atau simpul yang merupakan assignment dari ‘if’ diidentifikasi sebagai leftchild. Simpul ditengah yang merupakan assignment jika statement leftchild benar akan dimasukkan sebagai midchild. Sedangkan simpul paling kanan yang merupakan assignment jika leftchild salah diidentifikasi sebagai rightchild.

Setiap baris kemudian disimpan dalam tipe data array berdasarkan urutan dari kiri ke kanan. Misalkan untuk Gambar 8 akan menghasilkan urutan penyimpanan pohon sintaks sebagai berikut.

pohon_if = [>= id 60 writeln ‘Lulus’
writeln ‘Tidak Lulus’]

Pohon sintaks untuk setiap baris source code untuk Tabel 4.4 ditunjukkan pada Tabel 4.6. Jika setiap baris source code dapat menghasilkan pohon sintaks, dapat disimpulkan bahwa source code tersebut dapat dikompilasi. Untuk mengecek kesesuaian pohon sintaks yang dihasilkan dari source code mahasiswa dengan soal yang diberikan, maka akan dibandingkan dengan pohon sintaks dari source code jawaban. Jika sama maka dapat disimpulkan bahwa source code mahasiswa lolos uji kebenaran source code. Jika tidak sama, maka akan dihasilkan error berupa notifikasi bahwa baris source code tidak sesuai dengan soal.

Tabel 6 Pohon sintaks dari source code mahasiswa

Baris	Pohon sintaks
1	
2	
3	
4	
5	

4.3 Pengujian Sistem

Sistem penilaian source code otomatis yang telah dibangun kemudian diuji dengan menggunakan teknik acceptance testing yang

menitikberatkan pada kategori functionality dan performance. Pengujian dilakukan pada dua kelas Mata Kuliah Pemrograman Terstruktur (Pascal) dan dua kelas Mata Kuliah Pemrograman Java dengan dua responden dosen. Hasil pengujian sistem menggunakan teknik acceptance testing untuk kategori functionality ditunjukkan pada Tabel 7.

Tabel 7. Hasil acceptance testing untuk kategori functionality

Identifikasi Kebutuhan	Terpenuhi	Keterangan
Input source code mahasiswa melalui Browser	√	- Penjelasan terdapat pada halaman 37, 51
Input source code jawaban dosen melalui Browser	√	- Penjelasan terdapat pada halaman 37, 51
Dilakukan pengujian source code berdasarkan analisis leksikal	√	- Penjelasan terdapat pada halaman 37, 40
Dilakukan pengujian source code berdasarkan kompleksitas siklomatik dan independent path	√	- Penjelasan terdapat pada halaman 37, 43
Output sistem berupa daftar hasil analisis	√	- Penjelasan terdapat pada halaman 37, 53

Hasil pengujian teknik acceptance testing untuk kategori functionality yang terdapat pada Tabel 7 menunjukkan bahwa sistem penilaian source code otomatis bersifat generik telah memenuhi semua kebutuhan user sehingga tidak perlu dilakukan revisi terhadap sistem yang dibangun.

Tabel 8. Hasil acceptance testing untuk kategori performance

Contoh Tugas	Waktu Kerja Secara Manual				Waktu Kerja Sistem
	Responden 1		Responden 2		
Membuat total gaji	30	01:30:00	29	01:48:00	00:01:52
	36	01:45:00	35	01:50:00	00:02:04
Mengurutkan bilangan	30	01:35:00	29	01:40:00	00:02:10
	36	01:40:00	35	01:40:00	00:02:21
Menghitung pangkat dan faktorial	30	01:20:00	29	01:35:00	00:01:50
	36	01:30:00	35	01:41:00	00:01:55

Ket : angka ‘30’, ‘36’, ‘29’ dan ‘35’ adalah jumlah mahasiswa dalam kelas yang diujikan

waktu kerja sistem adalah waktu kerja rata-rata dari pemeriksaan kelas responden 1 dan 2

Dari hasil pengujian yang ditunjukkan pada Tabel 4.8 tampak bahwa Sistem Penilaian Source Code Otomatis Bersifat Generik membutuhkan waktu kerja yang jauh lebih singkat dibandingkan dengan waktu kerja manual. Sehingga dapat disimpulkan bahwa sistem ini lolos uji acceptance testing untuk kategori performance karena mampu meningkatkan efisiensi waktu yang dibutuhkan oleh responden untuk memeriksa source code mahasiswa.

5. PENUTUP

5.1 Kesimpulan

Sistem penilaian source code otomatis bersifat generik merupakan sistem yang dibuat dalam bentuk modul-modul yang bertujuan untuk memeriksa tugas program mahasiswa dalam bahasa pemrograman Pascal maupun Java.Sistem ini dibuat berdasarkan hasil identifikasi kebutuhan user, dalam hal ini dosen pengampu mata kuliah pemrograman terstruktur dan mahasiswa yang mengambil mata kuliah tersebut yang kemudian diimplementasikan ke dalam pembuatan sistem berdasarkan pendekatan berorientasi objek dengan tujuan agar dapat dikembangkan dengan mudah tanpa merusak sistem aslinya.

Aplikasi client/antarmuka untuk sistem penilaian source code otomatis bersifat generik adalah berbasis web, dengan tujuan agar sistem ini dapat berjalan pada berbagai platform yang mendukung aplikasi berbasis web.Aplikasi server untuk sistem ini dikembangkan dengan menggunakan bahasa pemrograman Java karena mendukung pengembangan berorientasi objek dan memiliki runtime yang cepat.

Sistem ini memeriksa source code mahasiswa melalui tiga tahap pengujian kebenaran source code, yaitu tahap analisis leksikal, tahap analisis flowgraph, dan tahap analisis sintaks. Dengan melalui ketiga tahap tersebut, diharapkan bahwa source code yang dianalisis dapat ditentukan kebenarannya meskipun tanpa dilakukan kompilasi.

Pengujian sistem dengan teknik acceptance testing baik kategori functionality maupun performance menunjukkan bahwa sistem ini dapat diharapkan untuk menguji kebenaran source code dan juga meningkatkan efisiensi pemeriksaan.

5.2 Saran

Sistem penilaian source code otomatis bersifat generik dikembangkan berdasarkan teknik kompilasi, meskipun masih terbatas pada tahap analisis leksikal dan analisis sintaks. Sistem ini dapat dikembangkan sehingga menjadi compiler yang utuh dan dapat memeriksa source code yang menggunakan bahasa pemrograman Pascal dan Java.Sistem juga dapat dikembangkan untuk memeriksa source code dalam bahasa pemrograman lainnya, seperti C++.

6. Referensi

[**Aaby, 2004]** Anthony A. Aaby (2004). Theory Introduction To Programming Languages.
<
http://www.cs.wwc.edu/~aabyan/Logic/index.html >

[Alfonseca, 2005] Enrique Alfonseca, Rosa M. Carro, Manuel Freire, Alvaro Ortigosa, Diana Pérez dan Pilar Rodriguez (2005). Authoring of Adaptive Computer Assisted Assessment of Free-text Answers. Dimuat pada jurnal Educational Technology & Society, Edisi 8 (3), h.53-65.
< http://www.ifets.info/journals/8_3/6.pdf >

[Blumenstein, 2004] Michael Blumenstein, et al (2004). An Experimental Analysis of GAME : A Generic Automated Marking Environment. Prosiding untuk ITiCSE 04, 2830 Juni 2004, Leeds, Inggris.

[Bonham, 2002] Scott W. Bonham, Duane L. Deardorff, Robert J. Beichner (2002). A comparison of student performance using web and paper-based homework in college-level physics. Dimuat pada Journal of Research in Science Teaching. Volume 40 Issue 10, Pages 1050 - 1071
<
http://www.ncsu.edu/per/Articles/HomeworkComparisonResearch.pdf >

[Carter, 2003] Janet Carter, et al (2003). How Shall We Assess This? Prosiding untuk ITiCSE 2003.

[Chi, 1998] Yu-Liang Chi, Philip M. Wolfe. (1998). A Web-Based Automatic Software Grading System. Prosiding untuk The 50th International Engineering Solution Conference,1999, Phoenix. <http://phoenix.mis.cycu.edu.tw/paper/1999-IEESC_Phoenix.pdf >

[Cooper, 2003] Keith Cooper, Linda Torczon (2003). Engineering A Compiler. Morgan Kaufman.

[Forsythe, 1965] George E. Forsythe, Niklaus Wirth (1965). Automatic Grading Programs. Dimuat pada Technical Report CS17, 12 Februari 1965, Universitas Stanford. < http://historical.ncstrl.org/litesite-data/stan/CS-TR-65-17.pdf >

[Grune, 1998] Dick Grune, Ceriel JH. Jacobs. 1998. Parsing Techniques : A Practical Guide. Amsterdam : Vrije Universiteit Amsterdam.

[Halide, 2010] Lidemar Halide. (2010). Autimatic Assessment untuk Mata Kuliah Bahasa Pemrograman pada Sistem Pembelajaran Berbasis Web. Makassar: Universitas Hasanuddin.

[Jackson, 1997] David Jackson, Michelle Usher (1997). Grading Student Programs using ASSYST. Prosiding untuk 28th SIGCSE Technical Symposium on Computer Science Education, San Jose, United States, 1997.

[Kumar, 2002] Amruth N. Kumar (2002). Learning Programs By Solving Problems. < http://phobos.ramapo.edu/~amruth/r/c/ictem/02/paper.pdf >

[MacLennan, 1983] Bruce J. MacLennan (1983). Principles of Programming Languages : Design, Evaluation and Implementation. New York: Holt-Saunders

[McCabe, 1976] Thomas McCabe (1976). A Complexity Measure. Dimuat pada IEEE Transactions on Software Engineering, Vol SE-2, No.4, Desember 1976.

[Mason, 2002] Oliver Mason, Ian Grove-Stephensen. (2002). Automated free-text marking with Paperless School. <http://magpie.lboro.ac.uk/dspace/bitstream/2134/1882/1/Mason_o1.pdf >

[Rani, 2010] Rani Purbaningtyas. (2010). Pengembangan Prototype Sistem Penilaian Otomatis Mata Kuliah Pemrograman Java. Makassar : Universitas Hasanuddin.

[Slamet, 1992] Sumantri Slamet dan Heru Suhartanto. (1992). Teknik Kompilasi. Jakarta : Universitas Indonesia.

[Slonneger, 1995] Kenneth Slonneger, Barry L Kurtz (1995). Formal Syntax and Semantics of Programming Languages. Menlo Park: Addison Wesley.

[Sultanoðlu, 1998] Sencer Sultanoðlu, Ümit Karakaþ (1998). Complexity Metrics and Models. < http://yunus.hacettepe.edu.tr/~sencer/complexity.html >

[Swartz, 2006] Fred Swartz (2006). Java : Complexity Measurement <http://www.leepoint.net/notes-java/principles_and_practices/complexity/complexity_measurement.html >

[Watt, 1990] David A. Watt (1990). Programming Language Concepts and Paradigms. New York: Prentice Hall.

[Zin, 2001] Abdullah Mohd Zin, Dr. Eric Foxley. (2001). Automatic Program Assessment System. <http://www.cs.nott.ac.uk/CourseMarker/more_info/html/ASQA.HTM >